

Chapter 2 · Pandas — DataFrame and Import/Export

CBSE Class 12 Informatics Practices (065) · Unit 1 = 25 marks · Ultimate Revision Sheet

60-SECOND RECALL

DataFrame	2-D labelled structure — rows and columns, heterogeneous , size mutable
loc vs iloc	loc = LABEL based, end INCLUDED · iloc = INTEGER based, end EXCLUDED
Add column	<code>df['new'] = values</code>
Delete column	<code>df.drop('col', axis=1)</code> or <code>del df['col']</code>
Delete row	<code>df.drop(index, axis=0)</code>
axis rule	axis=0 → ROWS · axis=1 → COLUMNS

1. Creating a DataFrame

SYNTAX

```
pd.DataFrame(data, index=None, columns=None)
```

THE FOUR WAYS 1. From a dictionary of LISTS — keys become column names

```
d = {'Name': ['Amit', 'Bina', 'Chetan'],  
     'Marks': [85, 92, 78]}  
df = pd.DataFrame(d)
```

OUTPUT

```
      Name  Marks  
0    Amit     85  
1    Bina     92  
2  Chetan     78
```

2. From a LIST OF DICTIONARIES — each dict is a row; missing keys become NaN

```
data = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]  
df = pd.DataFrame(data)
```

OUTPUT

```
      a  b  c  
0  1  2  NaN  
1  5 10 20.0
```

3. From a dictionary of SERIES — indexes are aligned

```
d = {'one': pd.Series([1,2,3], index=['a','b','c']),  
     'two': pd.Series([4,5,6,7], index=['a','b','c','d'])}  
df = pd.DataFrame(d)
```

OUTPUT

```
      one  two  
a  1.0  4  
b  2.0  5  
c  3.0  6  
d  NaN  7
```

4. From a 2-D ndarray / list of lists

```
df = pd.DataFrame([[1,2],[3,4]], columns=['A','B'])
```

2. DataFrame Attributes

Attribute	Returns
<code>df.index</code>	Row labels
<code>df.columns</code>	Column labels
<code>df.values</code>	Data as a 2-D ndarray
<code>df.dtypes</code>	Data type of each column
<code>df.shape</code>	Tuple (rows, columns)
<code>df.size</code>	rows × columns — total number of cells
<code>df.ndim</code>	2 — always, for a DataFrame
<code>df.T</code>	Transpose — rows become columns
<code>df.empty</code>	True if the DataFrame has no element

TRAP For a 3×4 DataFrame: `shape` is (3, 4), `size` is 12, `ndim` is 2. Students often give `size` as the number of rows.

3. Selecting Data — `loc` and `iloc`

One DataFrame, two ways to point at the same rows.

posn / label		Name	Marks	
0	0	Asha	91	<code>df.loc[0:2]</code> → 3 rows — end INCLUDED counts by the LABEL written in the index
1	1	Bala	78	
2	2	Chen	85	
				<code>df.iloc[0:2]</code> → 2 rows — end EXCLUDED counts by POSITION, like a normal Python slice

Same numbers, different answers. `loc` includes the end, `iloc` does not.

The *i* in `iloc` stands for **integer position** — and Python slices never include their end.

With a default 0,1,2 index the label and the position match — which is why the trap works.

If the index were 'a','b','c', `loc` would need 'a':'c' while `iloc` would still use 0:3.

THE MOST IMPORTANT DISTINCTION IN THE UNIT

Basis	<code>loc[]</code>	<code>iloc[]</code>
Based on	LABEL (name)	INTEGER position
End of slice	INCLUDED	EXCLUDED
Example	<code>df.loc['a':'c']</code> → a, b, c	<code>df.iloc[0:3]</code> → positions 0, 1, 2

```
df['Marks']           # one column, as a Series
df[['Name', 'Marks']] # multiple columns, as a DataFrame

df.loc[1]             # row with label 1
df.loc[0:2]           # rows 0,1,2 - END INCLUDED
df.iloc[0:2]          # rows 0,1 - END EXCLUDED

df.loc[0:2, 'Name']    # rows and a column together
df.iloc[0:2, 0:2]      # by position

df[df['Marks'] > 80]    # Boolean filtering
```

Given df with 3 rows: df.loc[0:2] gives 3 rows, df.iloc[0:2] gives 2 rows

4. Adding, Modifying and Deleting

COLUMNS

```
df['Grade'] = ['A', 'A+', 'B']      # ADD a new column
df['Marks'] = df['Marks'] + 5        # MODIFY an existing column

df.drop('Grade', axis=1)             # DELETE - returns a copy
df.drop('Grade', axis=1, inplace=True) # DELETE permanently
del df['Grade']                      # DELETE permanently

df.rename(columns={'Marks': 'Score'}, inplace=True)
```

ROWS

```
df.loc[3] = ['Deepa', 88]           # ADD a row by label
df.drop(1, axis=0, inplace=True)     # DELETE row with label 1
df.rename(index={0: 'first'}, inplace=True)
```

THE TWO TRAPS 1. **axis:** axis=0 means rows, axis=1 means columns. The default for drop() is axis=0, so dropping a column requires axis=1.

2. **inplace:** without inplace=True, drop() and rename() return a new DataFrame and the original is unchanged. This is the most common silent mistake.

5. Iteration

```
for col in df:                  # iterates over COLUMN NAMES
    print(col)

for index, row in df.iterrows(): # row by row
    print(index, row['Name'], row['Marks'])

for label, series in df.iteritems(): # column by column
    print(label, series.values)
```

TRAP iterrows() gives (index, row) where the row is a Series.

iteritems() gives (column name, column) where the column is a Series.

Confusing the two is a common output-prediction error.

6. Head, Tail and Descriptive Functions

```
df.head()      # first 5 rows
df.tail(3)     # last 3 rows
df.info()      # summary: dtypes, non-null counts
df.describe()  # count, mean, std, min, 25%, 50%, 75%, max
```

Function	Returns
df.max() df.min()	Column-wise maximum / minimum
df.sum() df.mean()	Column-wise total / average
df.count()	Non-null count per column
df.median() df.mode()	Median / most frequent value
df.quantile()	Quantile values
df.var() df.std()	Variance / standard deviation

Add `axis=1` to any of these to compute **row-wise** instead of column-wise.

7. Importing and Exporting CSV

THE TWO COMMANDS — guaranteed in the paper *Import (read from CSV into a DataFrame):*

```
df = pd.read_csv("marks.csv")

df = pd.read_csv("marks.csv", sep=";",
                  header=0,
                  names=['Name', 'Marks'],
                  index_col='Name')
```

Export (write a DataFrame to CSV):

```
df.to_csv("output.csv")

df.to_csv("output.csv", index=False, header=True)
```

Parameter	Meaning
sep	The separator; default is a comma
header=0	Use row 0 as column names. <code>header=None</code> means the file has no header row.
names=[...]	Supply your own column names
index_col	Which column to use as the row index
index=False	In <code>to_csv</code> , do not write the index column into the file

TRAP In `to_csv()` the default is `index=True`, so an extra unnamed column of 0,1,2 appears in the file. If the question says “without the index”, you **must** write `index=False`.

8. Predict the Output highest-frequency question type

```
Q1 df = pd.DataFrame({'A':[1,2,3], 'B':[4,5,6]})
print(df.shape, df.size, df.ndim)
```

```
Q2 print(df.loc[0:1])  # 2 rows (labels 0 and 1)
print(df.iloc[0:1])    # 1 row (position 0 only)
```

```
Q3 data = [{'a':1, 'b':2}, {'a':5, 'b':10, 'c':20}]
df = pd.DataFrame(data)
print(df)
```

```
Q4 df.drop('B', axis=1)
print(df.columns)
```

9. Task Triggers wording → which method?

If the question says...	Use
"add a new column Total"	<code>df['Total'] = ...</code>
"delete the column Grade permanently"	<code>df.drop('Grade', axis=1, inplace=True)</code>
"delete the row with index 2"	<code>df.drop(2, axis=0, inplace=True)</code>
"display rows 1 to 3 by label"	<code>df.loc[1:3]</code> — end included
"display the first two rows by position"	<code>df.iloc[0:2]</code>
"show students scoring above 80"	<code>df[df['Marks'] > 80]</code>
"rename the column Marks to Score"	<code>df.rename(columns={...}, inplace=True)</code>
"display a statistical summary"	<code>df.describe()</code>
"show the data types of columns"	<code>df.dtypes</code> or <code>df.info()</code>
"read data from a CSV file"	<code>pd.read_csv("file.csv")</code>
"write to CSV without the index"	<code>df.to_csv("f.csv", index=False)</code>
"interchange rows and columns"	<code>df.T</code>
"total number of cells"	<code>df.size</code>

MOST-ASKED BOARD QUESTIONS

- [3-4M] Create a DataFrame from the given data and perform the stated operations.
- [2-3M] Predict the output of the given DataFrame code.
- [2M] Differentiate between `loc` and `iloc`.
- [2M] Write code to import from / export to a CSV file.
- [1M] What does `axis=1` mean? → **Columns**
- [1M] Difference between `shape` and `size`.

THE HABIT THAT SCORES — Whenever a question says “**permanently**” or “**from the DataFrame**”, add `inplace=True`. Whenever it involves a column, add `axis=1`. Those two keywords account for most of the lost marks in this chapter.